程序说明
version 1.0.0

修订历史

日期	版本	作者	描述
2015.8.21	1.0.0	GA237	实现 Jumper 类基本功能
2015.8.21	1.0.1	GA237	增加一些图片便于说明

目录

1 程序功能

功能名	实现函(后三者为 VOID)	说明
无参数构造 Jumper	Jumper()	Jumper 默认颜色为 RED
有参数构造 Jumper	Jumper(Color JumperColor)	设置 Jumper 为指定颜色
反应	act()	满足跳跃条件则跳跃,满足移动条件则移动，否则转向
转向	turn()	向右转一个随机角度(45 度整数倍)
跳两格	jump()	Jumper 强制跳两格（无论前方是否有边界或障碍物）
移动	move()	Jumper 强制移动一格（无论前面是否有边界或障碍物）
跳跃判断	canJump()	判断是否满足跳跃条件（前方的第二个没有障碍物）
移动判断	canMove()	判断是否满足移动条件（前方的第一个没有障碍物）
可放置 Jumper 判断	canPutOntoLocation(Location next)	判断 Jumper 是否可以在这个格存在

特别说明，现规定 Jumper 的行为准则如下：（以下把除花朵以外的 Actor 称为障碍物）

1. 当 Jumper 前面的两个格子没有障碍物的时候 Jumper 直接到达第二格；

2. 当 Jumper 前面的第一个格子有障碍物，第二个格子没有障碍物的时候 Jumper 直接到达第二格；

3. 当 Jumper 前面的第一个格子没有障碍物，第二个格子有障碍物的时候 Jumper 只能到达第一格；

4. 当 Jumper 前面的两个格子都有障碍物的时候 Jumper 只能转向；

5. 当 Jumper 前面的的两个格子范围内有边界的时候 Jumper 最多只能移动到其边界，不能跳出去；

6. Jumper 的转向是随机的，角度为 45 的整数倍。

2 实现过程

借助 eclipse 开发此次项目。

导入 gridworld.jar，新建 Jumper.java 与 JumperRunner.java 两个文件。

根据题目要求实现 Jumper 类，借用 Bug 类来实现，所以很多与 Bug 类相似。我们把 Bug 类直接复制粘贴至 Jumper 类，然后修改参数，增加了 jump，canJump 函数来实现跳跃功能，然后依旧保留 Bug 类中只移动一步的 move，canMove。需要注意的是，要把 move 函数里面的生成花朵的相关语句删除掉，因为不要求 Jumper 在移动过的地方留下花朵。为了判断是否可以跳跃，是否可以移动，则借助 Bug 类中判断是否能在特定位置放置 Actor 类的语句，如下：

```
        Grid<Actor> gr = getGrid();
        if (gr == null) {
            return false;
        }
        if (!gr.isValid(next)) {
            return false;
        }
        Actor neighbor = gr.get(next);
        return (neighbor == null) || (neighbor instanceof Flower);
```

考虑到 canMove 和 canJump 都用到这段语句，为了使程序复用率更高些，减少冗余，特地把这个功能另写成函数 canPutOntoLocation(Location loc)，然后修改 act 函数，把 jump，canJump 加入判断中，act 函数的实现十分简单明了，如下图：

```
    /**
     * jumps if it can jump, move if it can move, turns otherwise.
     */
    public void act()
    {
        if (canJump()) {
            jump();
        }
        else if (canMove()) {
            move();
        }
        else {
            turn();
        }
    }
```

然后直接运行程序观察 Jumper 的移动情况是否达到预期效果。

为了进一步验证实现是否正确，使用 JUnit 进行单元测试，详情请参见测试报告。

然后使用 Sonar 检查代码质量.正确配置环境后，修改 sonar.projectName，java-module.sonar.projectBaseDir, sonar.projectKey 全为 src(如下图)

```
#required metadata
#projectKey项目的唯一标识，不能重复
sonar.projectKey=src
sonar.projectName=src
sonar.projectVersion=1.0
sonar.sourceEncoding=UTF-8
sonar.modules=java-module

#Java module
java-module.sonar.projectName=Java Module
java-module.sonar.language=java
#.表示projectBaseDir指定的目录
java-module.sonar.sources=.
java-module.sonar.projectBaseDir=src
```

即可把 src 里面的文件进行风格检测，然后根据实训要求以及 sonar 分析结果改进代码。

最后，借助 eclipse 把 build.xml 文件导出，修改路径为自己想要把文件放置的路径，然后把一些.jar 包文件放置进去即可实现自动编译。如下图是做的主要修改:

```
<project basedir="." default="build" name="GridWorld">
<property environment="env"/>
    <property name="junit.output.dir" value="junit"/>
<property name="debuglevel" value="source,lines,vars"/>
<property name="target" value="1.7"/>
<property name="source" value="1.7"/>
<path id="JUnit 4.libraryclasspath">
    <pathelement location="./package/junit.jar"/>
    <pathelement location="./package/org.hamcrest.core_1.3.0.v201303031735.jar"/>
</path>
<path id="GridWorld.classpath">
    <pathelement location="bin"/>
    <pathelement location="./package/gridworld.jar"/>
    <path refid="JUnit 4.libraryclasspath"/>
</path>
```

3 总结体会

Part3 小组项目要求明确，又有报告模板，所以小组分工很好做。看了问题和练习以后写起来其实很简单，只要参照 Bug 类即可扩展出我们这次程序。为了使代码复用性更强，依据原 Bug 类中的是否可放置 Actor 类（如下图），

```
Actor neighbor = gr.get(next);
return (neighbor == null) || (neighbor instanceof Flower);
```

在 Jumper 类里把它另外写出用作复用，以便在可以判断是否可跳跃两格的同时可以用来判断是否可以移动一格，然后之后在自己的程序里添加 jump，canJump 函数，然后修改 act() 函数，即在 act()函数里添加判断 canJump 即可。为此，小组项目的程序阶段就顺利完成。

由于程序是借助 eclipse 开发，所以 eclipse 自动生成的 build.xml 文件中的路径会自动关联 eclipse 安装目录，当要把它单独拿出来在命令行下运行，则要改参数，并且要把一些.jar 包文件导出到文件中，改完后，小组成员之间互相测试看是否可以运行，有个 1.7 版本以及 1.6 版本的值区别外，其他均可顺利进行，即最后打开命令行就可以愉快地 ant，然后 ant 一个对象即可实现想要运行的程序。

在写代码时要好好参照原有的程序，在认真思考原有程序的架构后再去扩展写自己的代码，这样会事半功倍，简单许多！小组分工也帮了很多忙，写程序，写测试，写报告，分工明确，完成任务变得高效许多，为此明确分工必不可少！